Q&A: Electronics Engineering for the People
LOCATION: Australia
PAGE: 22

PROJECT WRAP UP: DIY Color Identifier
LOCATION: United States
PAGE: 28

INSIGhT: Linux Software Development Tools
LOCATION: United States
PAGE: 64

CIRCUIT CELLAR

ThE WORLD'S SOURCE f OR EMBEDDED ELECTRONICS ENGINEERING INf ORMATION

APRIL 2012
ISSUE 261

EMBEDDED PROGRAMMING

Build a Capacitive-Touch Personal Amp

An MCU for Low-Power Applications

Mesh Networking with the SNAP OS

An Examination of Parallel Capacitors

Sigma-Delta Modifiers Explained

PLUS

Software Simplified

Automation Reduces Repetitive Tasks
// Automatic Code Generators
// Configuration Control Programs
// Testing, Verification, & Validation
// And More

04>
7 25274 75349 9
$7.50 U.S. ($8.50 Canada)
www.circuitcellar.com

Reprinted by permission. For subscription information, call 1.800.269.6301, or visit www.circuitcellar.com/subscriptions. Entire contents copyright ©2012 Circuit Cellar Inc. All rights reserved.

by Jeff Bachiochi (USA)

SNAP to It (Part 1)

Mesh Networking Simplified

Mesh networking relies on nodes to capture, distribute, and reproduce data. It is the glue that keeps the Internet and cellular towers connected. While there are many standard protocols available for localized networks (e.g., Wi-Fi, Bluetooth, or wireless USB), this article focuses on the SNAP network operating system created by Synapse Wireless.

The telephone has all but eliminated letter writing, the long-time de facto standard for keeping in touch with friends and families. Today, not only has this convenience gone wireless, but it is being used for far more than staying in contact with one another. Parents arm their kids with cell phones for peace of mind. Kids have a texting agility that is inherited from their video gaming skills. It's a sad commentary that there are far fewer "minutes" devoted to cell conversations than the senseless plethora of audio and video candy available.

When our Boy scout troop goes camping, we encourage the boys to leave all of their electronics at home. When traveling any distance, we use walkie-talkies for communication between vehicles, even though the adults have cell phones. The boys share a responsibility to keep tabs on every vehicle in the group and make sure no one takes an unexpected turn. (It amazes me how each GPS will calculate certain routes differently.) This convoy-style string can sometimes exceed the distance of a walkie-talkie, making the middle vehicles the only link between the front runner and the rear sweeper. Communication must be relayed (repeated) by one radio that has contact with those ahead and behind. The number of repeats or hops is related to how much the vehicles are spread out and the range of the radios.

So it is with low-power communications. Since transmission distance is a function of power output, low-power radios have a rather limited range. This can be a good thing if you wish to limit the interference you cause. It can be a bad thing if you need to span a considerable distance within a group of radios. When a radio must not only transmit and receive its own data, but also relay the data of others, it is being used as part of a mesh network. As one may expect, the strategies for this may be as simple as every radio repeating what it has heard, or as sophisticated as dynamic mapping of the most efficient route. The simplest strategies can create a throughput clogging uproar of activity as each receives and retransmits the same message. So, at least a little bit of smarts are required to cut this to an acceptable level.

MESH NETWORKS

You can think of the Internet as the world's largest wired mesh network. Information travels from one router (repeater/director) to the next until it reaches its destination. Your IP

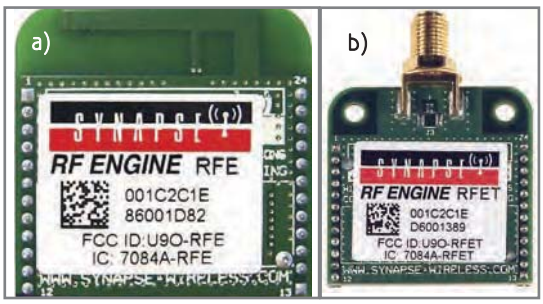


Photo 1—SNAP engine modules are available with an integrated antenna (**a**) and an RP-SMA connector for using an external antenna (**b**)

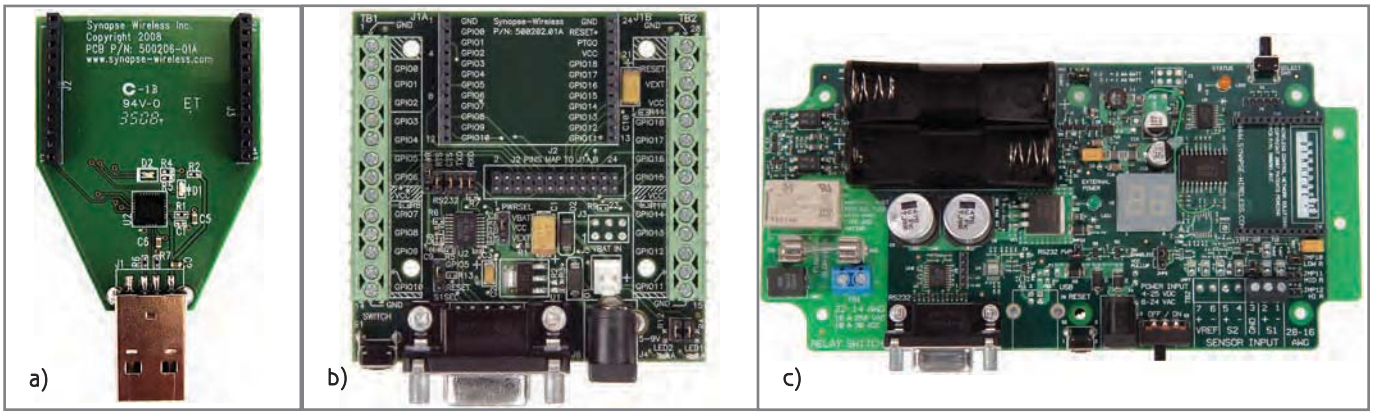


Photo 2—SNAP evaluation boards that accept all SNAP engine modules include the USB bridge module (**a**), the Prototype1 PCB (**b**), and the Prototype 2 PCB (**c**).

address changes when you disconnect your laptop and reconnect at some other location. (The IP is fixed to a physical connection.) To remain integrated in the mesh, when you move, you must reestablish your new location with others, either directly or through the grapevine.

On a smaller scale, if you work with microcontroller devices, you may be familiar with ZigBee mesh networks. ZigBee was conceived as the first low-power ad-hoc protocol in 2003, with the latest incarnation being ZigBee PRO. It should be noted that the ZigBee Alliance has recognized the need for simpler networks that remain within earshot (no repeaters) and has introduced ZigBee Radio Frequency for Consumer Electronics (RF4CE). A typical ZigBee network may consist of three types of devices, a ZigBee Controller (ZC), a ZigBee Router (ZR), and a ZigBee End Device (ZED). A single ZC originates and coordinates the network keeping track of all active nodes. ZRs can pass communication as a repeater and act as an endpoint for service, while ZEDs are strictly endpoints.

There are other protocols available you should consider when looking for something to use with your new widget. You may want to consider Wi-Fi, Bluetooth, or wireless USB as capable candidates depending on the application. Or, read on further as I take you through the SNAP network operating system (OS) created by Synapse Wireless.

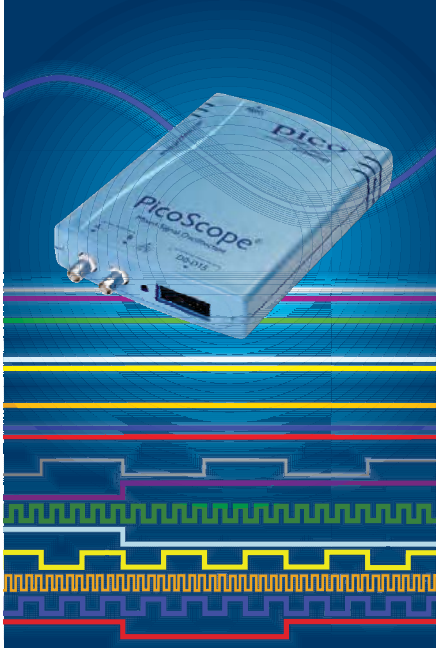
SYNAPSE WIRELESS

Not to be confused with USNAP (see my article, "Smart Network Access Point," *Circuit Cellar* 246, 2011), Synapse Wireless has created a family of products around the SNAP OS, which is a communication and control protocol used to create and maintain a mesh network of SNAP nodes. SNAP nodes contain a SNAP device (a microcontroller executing the SNAP OS) that interfaces with communication hardware (i.e., radio, I/O). A SNAP device can be integrated directly into your product, but is also available as part of a SNAP engine. The SNAP engine contains a SNAP device and radio prepackaged as a standard footprint module, providing a common development interface (see Photo 1).

A SNAP bridge is a SNAP node used to connect two networks such as a SNAP network and the Internet (SNAP Connect E10), a PC, or a second SNAP network. For instance, the SNAP Connect E10 is an end product incorporating a SNAP node as a bridge between a

ID 0	Reserved for Synapse use	ID 18	Collision avoidance	ID 40	SNAPPy CRC
ID 1	Reserved for Synapse use	ID 19	Radio unicast retries	ID 41	Platform
ID 2	MAC Address	ID 20	Mesh routing maximum timeout	ID 42–49	Reserved for future use
ID 3	Network ID	ID 21	Mesh routing minimum timeout	ID 50	Enable encryption
ID 4	Channel	ID 22	Mesh routing new timeout	ID 51	Encryption key
ID 5	Multicast processed groups	ID 23	Mesh routing used timeout	ID 52	Lockdown
ID 6	Multicast forwarded groups	ID 24	Mesh routing delete timeout	ID 53	Maximum loyalty
ID 7	Manufacturing date	ID 25	Mesh routing RREQ retries	ID 54–59	Reserved for future use
ID 8	Device name	ID 26	Mesh routing RREQ wait time	ID 60	Last version booted *(deprecated)*
ID 9	Last system error	ID 27	Mesh routing initial hop limit	ID 61	Reboots remaining
ID 10	Device type	ID 28	Mesh routing maximum hop limit	ID 62	Reserved for future use
ID 11	Feature bits	ID 29	Mesh sequence number	ID 63	Alternate radio trim value
ID 12	Default UART	ID 30	Mesh override	ID 64	Vendor-specific settings
ID 13	Buffering timeout	ID 31	Mesh routing LQ threshold	ID 65	Clock regulator
ID 14	Buffering threshold	ID 32	Mesh rejection LQ threshold	ID 66–127	Reserved for future use
ID 15	Intercharacter timeout	ID 33	Noise floor	ID 70	Transmit power limit
ID 16	Carrier sense	ID 34–38	Reserved for future use	ID 128–254	Available for user definition
ID 17	Collision detect	ID 39	Radio LQ threshold	ID 255	Reserved for Synapse use

Table 1—Nonvolatile memory is set aside for internal parameters that support function configuration. Additional slots are reserved for the user.

THE NEW PICOSCOPE
2205 MSO
MIXED SIGNAL OSCILLOSCOPE
GREAT VALUE, PORTABLE, HIGH END FEATURES AS STANDARD AND EASY TO USE


Think Logically...

pico Technology

Channels	2 Analog, 16 Digital
Resolution	8 bit
Bandwidth Digital frequency	Analog 25 MHz, Digital 100MHz combined
Sampling rate	200MS/s
Trigger modes	Edge, Window, Pulse width, Window pulse width, Dropout, Window dropout, Interval, Runt pulse, Digital, Logic
Price	$575

www.picotech.com/pco467
1-800-591-2796

```
Datamode.py

from synapse.switchboard import *

otherNodeAddr = "\x00\x00\xFF" # <= put the address of the OTHER node here

@setHook(HOOK_STARTUP)
def startupEvent():
    initUart(1, 9600) # <= put your desired baudrate here!
    flowControl(1, False) # <= set flow control to True or False as needed
    crossConnect(DS_UART1, DS_TRANSPARENT)
    ucastSerial(otherNodeAddr)
```

Figure 1—This simple script can be loaded into two nodes enabling them to act as a wireless serial link between two RS-232 devices.

ADC	
readAdc(channel)	Sample ADC on specified input channel, returns raw reading
CBUS Master Emulation	
cbusRd(numToRead)	Reads numToRead bytes from CBUS, returns string
cbusWr(byteStr)	Writes every byte in byteStr to the CBUS
GPIO	
setPinDir(pin, isOutput)	Set direction for parallel I/O pin
setPinPullup(pin, isEnabled)	Enable pull-up resistor for Input pin
setPinSlew(pin, isRateCon-	Enable slew rate control for Output pin
monitorPin(pin, isMonitored)	Enable GPIN events on Input pin
pulsePin(pin, msWidth, isPositive)	Apply pulse to Output pin
readPin(pin)	Read current level of pin
writePin(pin, isHigh)	Set Output pin level
setRate(rateCode)	Set pin sampling rate to off (0), 100 ms (1), 10 ms (2), or 1 ms (3)
I²C Master Emulation	
getI2cResult()	Returns the result of the most recent I²C operation
i2cInit(enablePullups)	Prepare for I²C operations
i2cRead(str, numBytes, retries, ignoreFirstAck)	Write str out, then read numBytes back in from I²C bus. Parameters retries and ignoreFirstAck are used with slow or special case devices
i2cWrite(str, retries, ignore-FirstAck)	Write str out over the I²C bus. Parameters retries and ignoreFirstAck are used with slow or special case devices
Miscellaneous	
setSegments(segments)	Set eval board LED segments (clockwise bitmask)
bist()	Built-in self test
eraseImage()	Erase user-application flash memory
resetVm()	Reset the embedded virtual machine (prep for upload)
initVm()	Initialize embedded virtual machine
vmStat(statusCode, args...)	Solicit a tellVmStat for system parameters
writeChunk(ofs, str)	Write string to user-application flash memory
chr(number)	Returns the character string representation of "number"
str(obj)	Returns the string representation of obj
int(obj)	Returns the integer representation of obj. Notice that you cannot specify the base. Decimal is assumed.
len(str)	Returns the length of string str (0–255)
random()	Returns a pseudo-random number 0–4,095
stdinMode(mode, echo)	Mode is 0 for line, 1 for character at a time

Table 2 continued on p. 71

Table 2—The built-in SNAPPy API functions available grouped together by the category.

Network	
getNetId()	Current network ID
setNetId(netId)	Set network ID (1–0xFFFE)
localAddr()	Local network address (3-byte binary string)
rpcSourceAddr()	Originating address of current RPC context (none if called outside RPC)
mcastSerial(dstGroups, ttl)	Set serial transparent mode to multicast
ucastSerial(dstAddr)	Set serial transparent mode to unicast
callback(callbackFnObj, remoteFnObj, args...)	RPC (back to the original invoker) of RPC results
callout(addr, callbackFnObj, remoteFnObj, args...)	RPC (to an arbitrary node address) of RPC results
rpc(dstAddr, remoteFnObj, args...)	RPC (unicast)
mcastRpc(dstGroups, ttl, remoteFnObj, args...)	RPC (multicast)
	Nonvolatile (NV) Parameters
loadNvParam(id)	Load indexed parameter from NV storage
saveNvParam(id, obj)	Save object to indexed NV storage location
	Radio
rx(isEnabled)	Enable/disable radio receiver
txPwr(power)	Adjust radio transmit level (0 is lowest, 17 is highest)
setChannel(channel)	Set radio channel
getChannel()	Radio channel
getLq()	Link quality in (–) dBm
getEnergy()	Detected RF energy in (–) dBm (current channel)
scanEnergy()	Detected RF energy in (–) dBm (all 16 channels)
peekRadio(addr)	Read a memory location from inside the radio
pokeRadio(addr, byteVal)	Write a memory location inside the radio

Table 2 continued from p. 70, continues on p. 72

SNAP network and a Linux environment running SNAP Connect. SNAP Connect is a licensed API that enables client applications to access the SNAP network using XML-RPC. XML-RPC is a standard communication protocol utilizing hypertext transfer protocol (HTTP) to transport remote procedure calls (RPCs) encoded using eXtensible Markup Language (XML).

SNAP Portal is a free PC application that turns your PC into a SNAP node enabling it to be part of the SNAP network through a bridge node connected via a serial or USB connection. I used a SNAP stick, a USB SNAP node dongle (see Photo 2), as this bridge. SNAP Portal is an incredible application tool that enables you to develop, download, and interact with every node on the SNAP network. Wait, if SNAP devices already handle all the networking tasks for me, what am I developing? While it's true you don't have to worry about the network stuff, you will most likely want to gather, disseminate, and make use of data. This requires application programming specific to your task at hand.

Mesh Networking, Simplified!

Experience the Fastest Growing Mesh Network Technology– SNAP® from Synapse Wireless.

Easiest platform for rapid prototyping and development.
Over-the-air programming with free IDE called *Portal.*
Batteries included for remote control applications.
Get ready for a world of Smart Machines.
Get SNAP!

SNAP Network Starter kit
Order yours today! MSRP $99.00
synapse-wireless.com/kit

SYNAPSE

Wireless Technology to Control and Monitor Anything from Anywhere™

© 2007-2012 Synapse Wireless Inc. All Rights Reserved. All Synapse products are patent pending.
500 Discovery Drive, Huntsville, Alabama · 877-982-7888

One of the most useful applications for a node is to become a SNAP sniffer. When a node is programmed as a SNAP sniffer, it monitors all network traffic and, when connected to a PC running the free sniffer application, you can see and log the data for debug purposes.

The SNAP OS handles all the networking tasks and provides the user with a Python-like virtual machine within each SNAP device. Your applications are written in this script language called SNAPPy (a subset of Python.) SNAPPy scripts are text files that are compiled into SNAPPy images to save space. You can create a single SNAPPy script for all nodes or create specialized scripts for each node. Before I get into the SNAPPy scripting language, let's look at mesh networks in general and see how SNAP handles different situations.

A MISHMASH OF MESHING

One feature of SNAP networks is that by design the protocol doesn't require any coordinator to keep track of everything. Each SNAP node can automatically retransmit a request to help reach nodes beyond the range of the sender. If every node was to do this, there could be a lot of unnecessary chatter. Therefore, it is important that SNAP recognizes who is not only within range, but who has the best shot at communicating with a neighbor (signal strength).

SNAPPy script functions that send data include parameters that adjust how a message is sent and retransmitted. These include the ability to isolate certain nodes by forming subgroups and defining how many times a message can be retransmitted.

Many functions are based on a configured parameter. SNAP uses nonvolatile memory to store these configurations between power cycles and resets. About half of the allocated space for these parameters is available for user-defined parameters (see Table 1). Network parameters can be adjusted to optimize how the nodes react, depending on your network's physical configuration.

SPI Master Emulation	
spiInit(cpol, cpha, isMsbFirst, isFourWire)	Setup for SPI with specified clock polarity, clock phase, bit order, and physical interface
spiRead(byteCount, bitsInLastByte)	Receive data in from SPI—returns response string (three-wire SPI only)
spiWrite(byteStr, bitsInLastByte)	Send data out SPI—bitsInLastByte defaults to 8, can be less
spiXfer(byteStr, bitsInLastByte)	Bidirectional SPI transfer—returns response string (four-wire SPI only)
Switchboard	
crossConnect(dataSrc1, dataSrc2)	Cross-connect SNAP datasources
uniConnect(dst, src)	Connect src->dst SNAP datasources
System	
getMs()	System ms tick (16 bit)
getInfo(which)	Get specified system information
getStat(which)	Get radio traffic status information
call()	Invoke a user-defined binary function
peek(addr)	Read a memory location
poke(addr, byteVal)	Write a memory location
errno()	Read and reset last error code
imageName()	Name of current SNAPPy image
random()	Returns a random number (0–4095)
reboot()	Reboot the device
sleep(mode, ticks)	Enter Sleep mode for specified number of ticks resolution, accuracy, and maximum duration vary between hardware platforms. For example, on an RF100 SNAP engine: In mode 0, ticks are 1.024 s each, ±30%. In mode 1, ticks are 1 s each, and can be 0–1073. On some platforms, negative values for ticks produce times shorter than 1 s.
UARTs	
initUart(uartNum, bps)	Enable UART at specified rate (zero rate to disable)
initUart(uartNum, bps, dataBits, parity, stop)	Enable UART at specified rate (zero rate to disable), data bits, parity, and stop bits
flowControl(uartNum, isEnabled)	Enable RTS/CTS flow control. If enabled, the CTS pin functions as a "Clear to Send" indicator
flowControl(uartNum, isEnabled, isTxEnable)	Enable RTS/CTS flow control. If enabled and parameter isTxEnable is True, then the CTS pin functions as a transmit enable (TXENA) signal. If enabled and isTxEnable is False, then the CTS pin functions as a "Clear to Send" indicator
	Table 2, continued from p. 71

SNAPPy

Table 2 shows the built-in functions that each node has available for use within SNAPPy scripts that a node will execute. The best way to learn how these are used is to look at the demos that are available for use. Many of these were designed for use with the SNAP prototyping or demo boards that form the basis of evaluation kits. These boards will accept the complete line of SNAP engines that plug in with the standard footprint. While two nodes form the minimum network, I will be using three nodes, one of which is the node used exclusively as a bridge from the PC to the network. A bridge node makes development much easier because you do not have to plug each node into the PC to program your SNAPPy script into it. This can be accomplished wirelessly using the bridge node.

Let's start with a simple script that enables two nodes to act as a wireless serial connection. This is a typical application for connecting two serial devices wirelessly (without a physical connection between them). One important note here is that there must be some way for nodes to identify their counterpart. A node is uniquely identified by its node address, which is the last 3 bytes of its MAC address printed on the label attached to each device (and pre-programmed into configuration register ID2.) The script in Figure 1 is the same for each node except that the node address of its counterpart is fixed within the script that is downloaded to a node. Let's go over this example script.

The first line imports the switchboard.py script from

the Synapse subdirectory. This script defines each data source (DS) object that can be used by the crossConnect() and uniConnect() functions. Here DS_UART1 and DS_TRANSPARENT are used. The second line defines the other node's address, which is used as the ucast destination for all outgoing communication. The third line indicates the beginning of an event, indicated by the @setHook() function. See Table 3 for a list of the available event hooks. This event is called upon reset and will run once. The user-defined function startupEvent() is all that is needed to complete this application. This function initializes UART1 for 9600 bps (note the databits, parity, and stopbits are not explicitly defined and so they default to 8, N, 1). Next the flow control for UART1 is disabled (CTS1 and RTS1 pins can be used as additional I/O). The crossConnect() function ties both inputs and outputs of one object to another object (here DS_UART1 and DS_TRANSPARENT are connected). All data received by UART1 is routed to the radio's transmitter and all data coming into the radio's receiver is routed to UART1's transmitter using transparent mode. Each radio can handle data as "transparent" or serial-packet (which is how RPCs are used). Finally, the ucastSerial() function limits transmissions to a single node. UcastSerial() has the advantage of being acknowledged by the recipient. An unacknowledged transmission will be resent based on the NV configuration ID 19 (Radio Unicast Retries). Assuming the edited script is loaded into two nodes, which have RS-232 level shifters, you can connect the nodes to serial devices and you're good to go. (Note: You could connect to the SNAP engine directly and use the TTL-level serial signals, where RS-232 levels are not needed.)

Since you have a number of NV ID reserved for your use, you may want to define one of these as the otherNodesAddr. This would enable the script to read this NV ID and use the address stored there as its counterpart. Why bother doing this? The ability to use a nonvolatile value enables it to be changed without having to rewrite the script using a hard coded address each time you want to pair up a set of nodes, as in the first example. Wouldn't it make sense to be able to pair SNAP nodes dynamically?

There are two ways of transmitting data. The first was just discussed, uniCastSerial(), the second is multiCastSerial(). When you send data using multicastSerial() it is sent as an unacknowledged message to all nodes. You can think of it as a general shout out. We've already seen that the user can define a function, as in the startupEvent() of the first example.

Suppose I defined a checkForNodes() function consisting of mcastRpc(1, 1, "whoIsOutThere"). This RPC would be multicast to all nodes (in group1 (all), using 1 hop, looking to invoke the function whoIsOutThere() on all those nodes listening. If the whoIsOutThere() function is defined consisting of the RPC function, rpc(rpcSourceAddr(), "iAm()"), then this enables the node to respond to the one who originated the request with a unicast RPC requesting to invode function "iAm(). The originator's function "iAm() consisted of otherNodesAddr = rpc(SourceAddr()), the variable otherNodesAddr would then hold the node ID of the first responding node. Finally, if the originator unicast back the rpc(rpcSourceAddr(), "iAm()") function, the responder could complete transaction knowing it was chosen as the pairing node. While this description is a bit simplistic, I hope you can see how one node may discover another node and swap node addresses by a mix of multicast and unicast transmissions.

Here are a few things you must think about for this to work. Assuming all nodes are using the same script with all of the above functions, there must be a way of determining when these functions should be allowed. What if there are more than two nodes out there? If a paired set of nodes has already saved the NV ID defined as otherNodesAddr, you could skip all of this pairing. If a findNewNodeAddress push button is implemented using @setHook(HOOK_GPIN), then a function might clear the NV ID otherNodesAddr. Any nodes with an erased address would be

BEST SCOPE SELECTION
& lowest prices!
25MHz Scope
Remarkable low cost 25MHz 2 channel plus trigger USB bench scope with 8 inch full color LCD display. Spectrum analysis and autoscale functions.
PDS5022S $279
iPhone Scope
5MHz mixed signal oscilloscope adapter for the iPhone, iPad and iPod Touch! A FREE iMSO-104 app is available for download from Apple App Store.
iMSO-104 $297.99
100MHz Scope
High-end 100MHz 1GSa/s 2-channel benchscope with 1MSa memory and USB memory port. Includes a FREE scope carry case! New low price!
DS1102E $399
60MHz Scope
60MHz 2 channel digital scope with a 500 MSa/s sample rate, 10MSa memory & 8" color TFT-LCD screen.
SDS6062 $399
World's Smallest
The world's smallest MSO! This DIP-sized 200kHz 2 channel scope includes a spectrum analyzer and arbitrary waveform generator. It measures only 1 x 1.6 inches in size!
Xprotolab $49
100MHz MSO
2 channel 100MSa/s oscilloscope and 8-ch logic analyzer USB 2.0 and 4M samples storage per channel with sophisticated triggering and math functions.
CS328A $1359
Handheld 20MHz
Fast, accurate handheld 20MHz 1-ch oscilloscope.
- 100 M/S sample rate
- 3.5 in. color TFT-LCD
- 6 hour battery life
Inc. rugged impact-resistant case
HDS1021M $299.95
More selections at:
www.saelig.com
Follow us on:
Saelig .com
unique electronics

Hook Name	When Invoked	Parameters	Sample Signature
HOOK_STARTUP	Called on device bootup	HOOK_STARTUP passes no parameters.	setHook(HOOK_STARTUP) def onBoot() : pass
HOOK_GPIN	Called on transition of a monitored hardware pin	**pinNum** – The pin number of the pin that has transitioned.1 isSet – A Boolean value indicating whether the pin is set.	setHook(HOOK_GPIN) def pinChg(pinNum, isSet) : pass
HOOK_1MS	Called every millisecond	**tick** – A rolling 16-bit integer incremented every millisecond indicating the current count on the internal clock. The same counter is used for all four timing hooks.	setHook(HOOK_1MS) def doEvery1ms(tick) : pass
HOOK_10MS	Called every 10 ms	**tick** – A rolling 16-bit integer incremented every millisecond indicating the current count on the internal clock. The same counter is used for all four timing hooks.	setHook(HOOK_10MS) def doEvery10ms(tick) : pass
HOOK_100MS	Called every 100 ms	**tick** – A rolling 16-bit integer incremented every millisecond indicating the current count on the internal clock. The same counter is used for all four timing hooks.	setHook(HOOK_100MS) def doEvery100ms(tick) : pass
HOOK_1S	Called every second	**tick** – A rolling 16-bit integer incremented every millisecond indicating the current count on the internal clock. The same counter is used for all four timing hooks.	setHook(HOOK_1S) def doEverySec(tick) : pass
HOOK_STDIN	Called when "user input" data is received	**data** – A data buffer containing one or more received characters.	setHook(HOOK_STDIN) def getInput(data) : pass
HOOK_STDOUT	Called when "user output" data is sent	**HOOK_STDOUT** passes no parameters.	setHook(HOOK_STDOUT) def printed() : pass
HOOK_RPC_SENT	Called when the buffer for an outgoing RPC call is cleared	**bufRef** – an integer reference to the packet that the RPC call attempted to send. This integer will correspond to the value returned from getInfo(9) when called immediately after an RPC call is made. The receipt of a value from HOOK_RPC_SENT does not necessarily indicate that the packet was sent and received successfully. It is an indication that SNAP has completed processing the packet.	setHook(HOOK_RPC_SENT) def rpcDone (bufRef) : pass

Table 3—SNAP is an event-driven language. These functions are built-in events that a SNAPPy script can use within its application.

looking for a pairing request by enabling the whoIsOutThere() function. A second push button, "iAmLooking" could invoke the whoIsOutThere() function using @setHook(HOOK_GPIN). The first responder would get paired.

Another approach might be to use the node's serial connection and a terminal program. A command-line interpreter script would enable you to request and collect responding node addresses, display, and choose from among the available nodes. How does a command-line interpreter relate to a node that is being used as a transparent serial link? The transparent application just shuffles data from endpoint to endpoint (the UART at each end of the wireless link). A command-line interpreter will recognize specific data to initiate some event. In the crossConnect(DS_UART1, DS_TRANSPARENT) of the first example, there is no way for any of the data passing through to be analyzed. If you use crossConnect(DS_UART1, DS_STDIO) then you are connecting user I/O to UART1. Data received by the UART is available through the event @setHook(HOOK_STDOUT). Any function can send data to the UART for transmission via the print statement. Data is available as characters (Mode=1) or carriage return (CR) terminated lines (Mode=0) using stdin Mode(Mode, Echo). Echo=1 will resend the characters back to the sender. With this setup you can monitor the data and look for a specific string (like the "~~~" used in systems to provide an "AT" command set) to trigger a redirection of data. This might be the checkForNodes() function described earlier, after which data would again become "transparent."

If you followed this, you may be thinking transparent with what? There is no connection now to DS_TRANSPARENT and the radio? That is correct and it is illegal to connect a source, like the UART1 with two destinations like DS_STDIO and DS_TRANSPARENT. However, we can use the Rpc(address, function, arguments) statement to request the execution of a user function which could pass any data as the argument of that function. On the far end, that function can perform any necessary dissection of the data or merely use the print statement to pass it onto the UART. In this case, the UART and the STDIO objects are cross connected and the radio "connection" is made through RPC functions of the SNAPPy API.

LOOK MOM, NO HANDS

Most of the discussion so far has been on wirelessly

connecting two UART objects. It's an activity that most can relate to. What happens when the object is not a UART? The talk of connecting the UART object to the TRANSPARENT object makes a good mental picture, however there are other objects that don't require a connection at all. These are all SNAPPy API functions and can be used at any time. Referring back to Table 1 you can see that the scripts can contain functions that communicate with external CBUS, SPI, and I²C slave devices physically attached to predefined I/O pins. Internally, there are GPIO, digital I/O pins for logic monitoring and control, as well as an analog-to-digital converter (ADC), for converting analog inputs. Let's consider an example that uses only GPIO.

The simplest example I can think of might be a wireless doorbell. One node has a push button connected to a digital input pin and the other node has a piezo element connected to a digital output pin. If the push-button event uses a multicast function like `mcastRcp(group, ttl, function, arguments)`, then it doesn't even need to know where it is sending the message. The function will execute on any node that hears it. If the second node has its piezo element connected to the GPIO associated with pulse-width modulation (PWM), then the script can simply beep or, if you're clever, it can play a little ditty. You could simply add more push-button nodes for the other doors. In fact, each door button could request a different tone pattern.

To eliminate having to program each button node with a different script (requesting a different "ring tone") you can do this by using multiple inputs. You might create multiple button inputs, with each input associated with a particular value. Or, you might assign one input for the button and other inputs to configuration jumpers that the script can read as a selected value. In either case, the value is sent as the function's argument, defining which bell has been requested.

It can be noted that additional bell nodes could be placed in every room of the house all happy to execute their annunciation when requested by any door. Suppose the piezo element and associated transistor driver were not installed on the node? The SNAP device doesn't care, in fact it doesn't even know! So we could do additional things on this node like flash a light on and off, simply by using an unused I/O pin to drive any support circuitry needed for the task and include the script statements necessary to support that task. This way any bell node could have a different arrangement of circuitry and still use the same node script.

SPREADING THE WEALTH

Perhaps you can begin to see the possibilities with this type of network where everyone is created equal. Every node has the potential to act on data received from any other node in the network. Imagine your spouse complaining that the room lighting is a safety hazard at night. You could jump onto the network and tell the reading lamp to pay attention to the hall light switch and *voila*, an extra light now goes on and off with the hall switch.

I looked briefly at using a command-line interpreter with a SNAP node, which can give your PC applications the ability to interact with your scripts via a serial connection. Optionally, the licensed SNAP connect server application will enable your applications (written in almost any language) to directly invoke functions on any SNAP node, even over the Internet, using XML-RPC.

The term machine-to-machine (M2M) is becoming a popular phrase today as it refers to the technologies that enable systems to communicate with one another. You know things are getting serious by the number of standards committees that have been formed. This just may be one market where we can flex our technological muscles. And, if security is an issue for you, each SNAP device has a Basic encryption algorithm built in. AES-128 encryption is also available on most devices by simply downloading a special version of the SNAP OS. It is good programming practice to be confident in your application before turning on any encryption, as debugging and use of the sniffer will be all but futile.

Jeff Bachiochi (pronounced BAH-key-AH-key) has been writing for Circuit Cellar *since 1988. His background includes product design and manufacturing. You can reach him at jeff.bachiochi@imaginethatnow.com or at www.imaginethatnow.com.*

RESOURCES

BiTXml, www.bitxml.org.

Connected World magazine, www.connectedworldmag.com.

M2M magazine, www.machinetomachinemagazine.com.

M2MXML, m2mxml.sourceforge.net.

Synapse Wireless, Inc., "SNAP Network Operating System Reference Manual for Version 2.4," 2011, http://forums.synapse-wireless.com/upload/SNAP%20Reference%20Manual.pdf.

———, "Technical Manual SNAP Hardware Document Revision v1.3," 2010, http://forums.synapse-wireless.com/upload/SNAP%20Hardware%20Technical%20Manual.pdf.

Telecommunications Industry Association, "TR-50 Smart Device Communications," www.tiaonline.org/standards/committees/committee.cfm?comm=tr-50.

———, "M2M Standardization Task Force (MSTF)," www.tiaonline.org/standards/mstf/index.cfm.

ZigBee Alliance, "ZigBee RF4CE Overview," 2009, www.zigbee.org/Specifications/ZigBeeRF4CE/Overview.aspx.

SOURCE

SNAP Connect E10 embedded Linux computer
Synapse Wireless, Inc. | www.synapse-wireless.com

CIRCUIT CELLAR

Subscribe Now!

Subscribe now to the leading computer applications magazine specializing in embedded systems and design!

12 issues per year for just

Print OR Digital: **$50** :: Combo (Print + Digital): **$85**

www.circuitcellar.com/subscription